GRAPH TYPES FOR MONADIC MOBILE PROCESSES(Concurrency Theory and Applications '96)

AUTHOR(S):

YOSHIDA, NOBUKO

CITATION:

YOSHIDA, NOBUKO. GRAPH TYPES FOR MONADIC MOBILE PROCESSES(Concurrency Theory and Applications '96). 数理解析研究所講究録 1997, 996: 158-161

ISSUE DATE:

1997-05

URL:

http://hdl.handle.net/2433/61232

数理解析研究所講究録
996巻 1997年 158-161

GRAPH TYPES FOR MONADIC MOBILE PROCESSES — EXTENDED ABSTRACT —

NOBUKO YOSHIDA

The monadic π-calculus [13, 11, 5, 3, 6] is a powerful formalism in which we can construct complex structures of concurrent computing by combining simple monadic name passing. The construction of significant computational structures in this calculus is done by passing and using private names between interacting parties to control the sharing of interaction points. For example, the following process expresses communication of a sequence of names (below $ax.P$ is input and $\overline{a}v.P$ is output, $(a)P$ denotes scope restriction, and c, z are fresh).

$$az.zx_1.zx_2.zx_3.P \mid (c)\ \overline{a}c.\overline{c}v_1.\overline{c}v_2.\overline{c}v_3.Q \longrightarrow\!\!\!\!\rightarrow\ P\{v_1v_2v_3/x_1x_2x_3\} \mid Q \tag{0.1}$$

In this example, coming from [13], the private channel c is used during interaction, so that, after the initial step, the communication of v_1, v_2 and v_3 is done deterministically without interference from the third party. This example also shows that we can represent polyadic (multiple) name passing from monadic name passing. Another example with a more complex communication structure follows.

$$az.zx_1.\overline{z}v_1.zx_2.\overline{z}v_2.P \mid (c)\ \overline{a}c.\overline{c}w_1.cy_1.\overline{c}w_2.cy_2.Q \longrightarrow\!\!\!\!\rightarrow\ P\{w_1w_2/x_1x_2\} \mid Q\{v_1v_2/y_1y_2\} \tag{0.2}$$

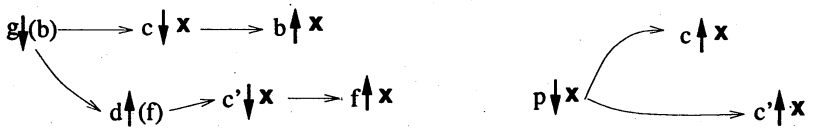
Note input and output are mixed together. As in the previous example, once two parties start interaction, a sequence of communication is done following a prescribed protocol using a private channel. The same scheme can be easily generalised to more complex communication structures, including those with parallelism and complex information flow.

As a means to study rich computational structures representable in name passing processes, a notion of types called *sorting* was introduced by Milner [12] and has been studied extensively since then [1, 19, 2, 16, 17, 10, 18, 4, 9]. Sorting shows how a name carries another name. For example, if v has a type, say, **nat**, and we have a term $\overline{a}v.\mathbf{0}$, then a should have a type (**nat**), a type which carries **nat**. This idea and its ramifications have been used to analyse significant semantic properties of calculi with polyadic name passing. However, as was already noticed in [12], the sorting in the monadic π-calculus is not powerful enough to type-abstract known encodings of polyadic name passing like (0.1) above: Indeed, (0.1) becomes ill-sorted if v_1, v_2, v_3 have different sorts. As far as we know, the situation remains the same with the

Department of Computer Science, University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ, UK., e-mail: ny@dcs.ed.ac.uk. tel: 44-131-650-4889. fax: 44-131-667-7209.

refinements of sorting proposed in [16, 9, 4]. This means, among other things, the sorting restricted to the monadic terms does not give as rich semantic analysis as in the polyadic case, while behaviourally the above encoding does mimic polyadic name passing. In general, this is because the sorting does not capture the dynamic structure of interaction, especially those using the transmission of private names as in (0.1) and (0.2), which is omnipresent even in the polyadic setting.

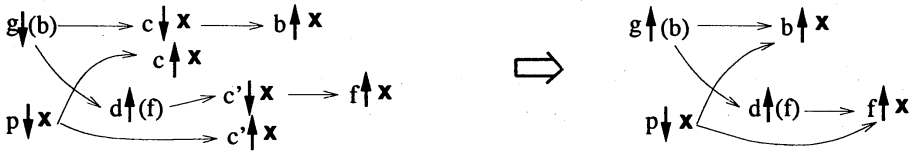
In [22], we develop a syntactic theory of types for monadic π-calculus which extends the sorting in a simple way and by which we can extract the abstract operational structures of π-terms including those of (0.1), (0.2) as well as more complex ones (cf.[22]). The key technical idea is to represent a class of dynamic communication behaviour of mobile processes using simple combinatorial expressions, i.e. graphs. In a graph type, nodes denote atomic actions and edges denote the activation ordering between them, representing as a whole a deterministic protocol a process will be engaged in. As a simple example, suppose we have two terms $gb.(cx.\overline{b}x \mid (f)\overline{d}f.c'x.\overline{f}x)$ and $px.(\overline{c}x \mid \overline{c'}x)$. Then they may be given types: Here on the left side, the action



$g\downarrow(b)$ (the input of fresh name b) should take place directly preceding $c\downarrow\mathbf{x}$ and $d\uparrow(f)$ (and indirectly all the rest), while there is no activation ordering between, for example, $c\downarrow\mathbf{x}$ and $f\uparrow\mathbf{x}$ in the same graph. Now if two protocols are compatible, we can compose them via "cuts," i.e. pairs of complementary atomic actions, to yield more complex types. Think of the following composition of the above two terms:

$$(cc'f)\ (\ \overline{g}b.(cx.\overline{b}x \mid \overline{d}f.c'x.\overline{f}x) \mid px.(\overline{c}x \mid \overline{c'}x)\ ) \tag{0.3}$$

The process of giving a type to this composed term is illustrated in the following.



The picture shows how the term (0.3) is given a type on the right hand side after the procedure of "cut elimination." Notice how the original activation ordering relations are merged to generate a new relation, so that a proper graph structure arises even if the original types are trees.[1] In the above example, the existence of two arrows going to $b\uparrow\mathbf{x}$ shows that the action should take place after its two parent actions take place. It also shows how the graph-based representation allows a refined expression

[1]We also note that such protocol composition enables the distribution and merging of information in communication, and the increase of parallelism, so can have a practical significance. It also plays a key role in theory of combinators for mobile processes [6, 7, 21].

of the flow of control in comparison with the syntactic construct like prefix, while still keeping a simple combinatorial structure. Compared with sorting and its refinements, the graph type tries to capture dynamic interactive behaviour of name passing processes including the new name passing (as $d\uparrow(f)$ above) based on a simple graphical expression.

This departure from the type abstraction of static usage of names to that of dynamic process behaviour makes it possible to type-abstract many non-trivial computational structures representable in π-calculus. Indeed it allows us to solve Milner's open issue mentioned above in a sharper form. We show that not only the encoding of sorted polyadic π-terms into the monadic terms is typable preserving the original type structure, but also it results in *equational full abstraction*: let $=_{\mathrm{P}}$ and $=_{\pi}$ be suitably defined behavioural equalities over sorted polyadic π-terms and monadic π-terms, respectively. Then we get:

$$P =_{\mathrm{P}} Q \;\Leftrightarrow\; [\![P]\!] =_{\pi} [\![Q]\!]$$

where $[\![\cdot]\!]$ is the standard encoding from polyadic π-terms to monadic π-terms. In the untyped setting, we can only get "$\Leftarrow$" (adequacy) direction in terms of usual weak behavioural equivalences. This is due to possible violation of "protocols" by environment processes, and relates to the lack of precise type abstraction of the communication structure of various encodings such as (0.1) and (0.2) in preceding type disciplines, even though the effect of process types on behavioural equalities has been studied since the work of Pierce and Sangiorgi [16, 9]. In this context, our result (which seems the first of this kind with respect to the encoding of polyadic name passing) shows how precisely our type discipline captures essential behavioural information these protocols carry. What may be surprising is that this can be done using a simple idea of graph-based types and a small typing system. Moreover the results can be easily extended to encodings of calculi with more complex communication structures, cf.[22]. We also notice that the polyadic name passing is a typical example of encoding of high-level communication structures into π-calculus, cf. [11, 8, 20], so that the presented construction and its extensions will hopefully become a basis for using typed equality over π-terms to verify semantic equality of various target languages and calculi in a uniform framework. The detailed definitions of graph types, the full abstraction result shown in this abstract and their proofs can be found in [22].

References

[1] Gay, S., A Sort Inference Algorithm for the Polyadic π-Calculus. *POPL'93*, ACM Press, 1993.

[2] Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, pp.509–523, Springer-Verlag, 1993.

[3] Honda, K., *A Study of ν-calculus and its Combinatory Representation*, Phd Thesis in the Department of Computer Science, October, 1994.

[4] Honda, K., Composing Processes, *POPL'96*, pp.344-357, ACM Press, 1996.

[5] Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, pp.133–147, Springer-Verlag 1991.

[6] Honda, K. and Yoshida, N., Combinatory Representation of Mobile Processes, *POPL'94*, pp.348–360, ACM Press, 1994.

[7] Honda, K. and Yoshida, N., Replication in Concurrent Combinators, *TACS'94*, LNCS 789, pp.786–805, Springer, 1994.
[8] Jones, C.B., *Process-Algebraic Foundations for an Object-Based Design Notation.* UMCS-93-10-1, Computer Science Department, Manchester University, 1993.
[9] Kobayashi, N., Pierce, B., and Turner, D., Linear Types and π-calculus, *POPL'96*, pp.358–371, ACM Press, 1996.
[10] Liu, X. and Walker, D., A polymorphic type system for the polyadic pi-calculus, CONCUR'95, LNCS, Springer-Verlag, 1995.
[11] Milner, R., Functions as Processes. *Mathematical Structure in Computer Science*, 2(2), pp.119–146, 1992.
[12] Milner, R., Polyadic π-Calculus: a tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1992.
[13] Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), pp.1–77, 1992.
[14] Odersky, M., Applying π: Towards a basis for concurrent imperative programming, *2nd. SIPL*, pp.95–108, 1995.
[15] Odersky, M., Polized Name Passing. *FSTTCS'15*, LNCS, Springer-Verlag, 1995.
[16] Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *LICS'93*, pp.187–215, 1993.
[17] Takeuchi, K., Honda, K. and Kubo, M., An Interaction-based Language and its Typing System. *PARLE'94*, LNCS 817, pp.398–413, Springer-Verlag, 1994.
[18] Turner, D., The π-calculus: Types, polymorphism and implementation, Phd Thesis, University of Edinburgh, 1996.
[19] Vasconcelos, V. and Honda, K., Principal Typing Scheme for Polyadic π-Calculus. *CONCUR'93*, LNCS 715, pp.524-538, Springer-Verlag, 1993.
[20] Walker, D., Objects in the π-calculus. *Information and Computation*, Vol. 116, pp.253–271, 1995.
[21] Yoshida, N., Graph Notation for Concurrent Combinators, *TPPP'94*, LNCS 907, pp.393–412, Springer-Verlag, 1995.
[22] Yoshida, N., Graph Types for Mobile Process Calculi, To appear in Proc. 16th FST/TCS'16, LNCS, Springer-Verlag, India, December, 1996. The full version is CS technical report, Keio University, 1996. Available from `ftp.dcs.ed.ac.uk/export/ny/pub/graph1.ps.Z`.